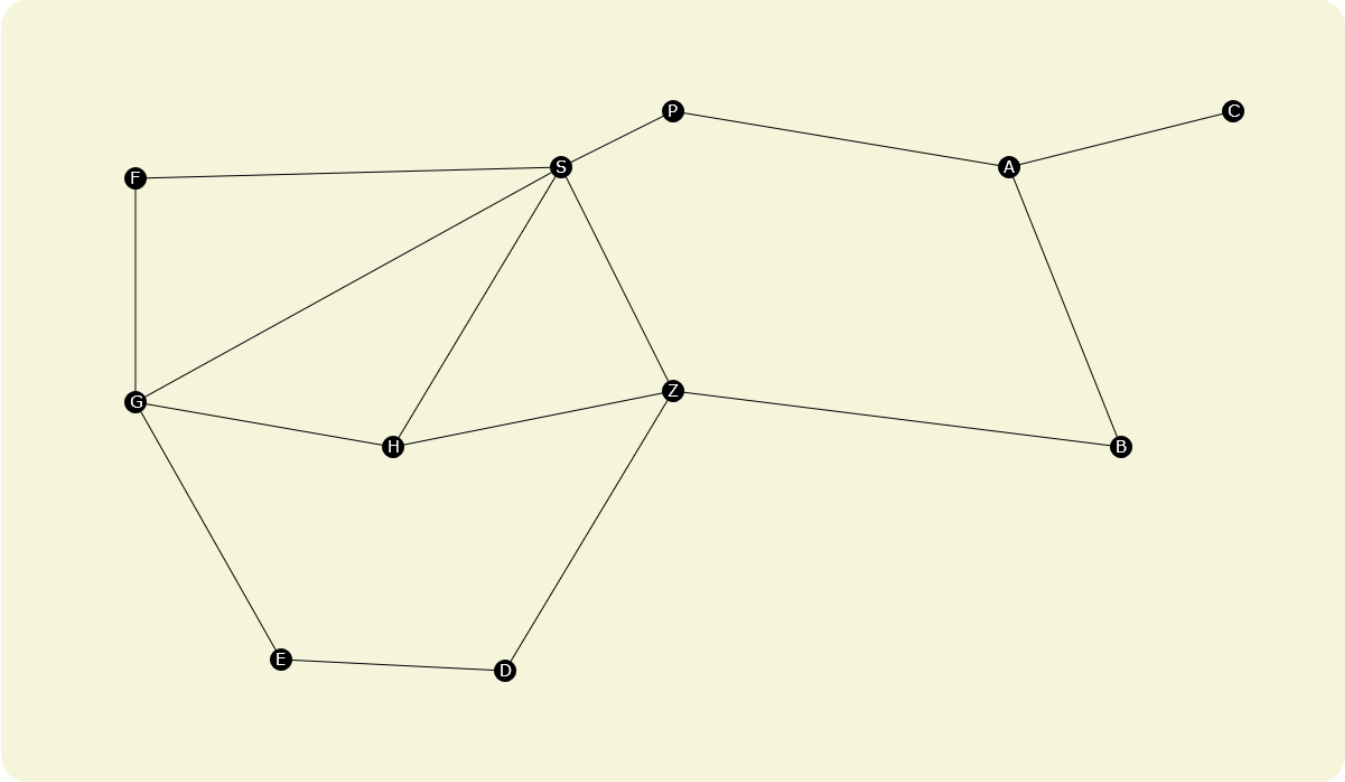
Pakkerobotten



I dette projekt betragtes en by med 11 lokationer forbundet med veje som skitseret på billedet.

Formål

Simulation af en automatiseret pakkerobot der opsamler og leverer pakker.

Krav

- De 11 lokationer og forbindelser imellem dem skal visualiseres.
- Automatisering af pakkerobotten skal implementeres.

Gruppearbejde: 2-4 personer.

Præsentation

Projektet afsluttes med en mundtlig præsentation i uge 39 på ca. 5-10 min for hver gruppe hvor følgende gennemgås:

- **Implementation:** Lille bid af koden fra den seneste version.
- **Test og Evaluering:** Demonstration af programmet og gerne et eksempel på noget der ikke virker.
- **Analyse og Design:** Hvad ville kravene til næste version være og hvordan kunne det struktureres?

Forslag til fremgangsmåde

I mappen `02a_Pakkerobotten` er der filer som man kan bruge som udgangspunkt. Udover HTML- og CSS-filen er der 4 JavaScript-filer. Man må selvfølgelig også gerne lave sit eget program fra bunden.

- data.js indeholder positioner (til et lærred på 1200x700px) og forbindelser mellem lokationer.
- visual.js indeholder et udgangspunkt til første visualisering.
- town.js indeholder udgangspunkter til struktureringen af lokationer.
- main.js sørger for at skabe en variabel til canvas i HTML der kan bruges til at tegne med og kan anvendes til at kalde funktioner i andre filer.

Generelle tips

- Under arbejdet med projektet kan det være en stor fordel at lave flere JavaScript-filer der hver indeholder sine funktionaliteter. Hvis man har hele koden i én fil bliver det hurtigt uoverskueligt.
- Man kan godt udvikle forskellige funktionaliteter i programmer sideløbende med hinanden. Alle i gruppen behøver altså ikke at arbejde på samme del af projektet samtidig. Dette kræver dog at man har en god ide om hvordan der "interfaces", dvs. hvilket input man kan forvente til sin del og hvilket output man skal levere.
- Hvis man sidder helt fast, kan man få noget inspiration ved at se på hvordan problemet er løst i bogens kapitel 7. Husk dog at man lærer meget lidt ved at kopiere direkte.

Visualisering

En nem måde at tegne på hjemmesiden vha. JavaScript er ved at bruge canvas. Se også referencen for canvas 2D context. I udgangspunktet bliver der i visual.js tegnet cirkler for alle lokationer, men på tilfældige steder.

- Tilpas funktionen `drawLocations()` i visual.js, så placeringen passer med koordinaterne angivet i objektet `coords` som er defineret i data.js. Angiv bogstavet for hver lokation.
- Få funktionen `drawRoads()`i visual.js til at tegne streger mellem alle lokationer der er forbundet, som defineret i array `roads` (også defineret i data.js).
- **Ekstra:** Tilføj funktioner der kan bruges til at markere robottens placering og hvor mange pakker der er ved forskellige lokationer. Det kan bl.a. gøres ved at tegne på canvas eller ved at oprette html-elementer der flyttes eller manipuleres i JavaScript.

Pakkerobot

- Man får brug for at kunne tjekke hvilke andre lokationer som et sted er forbundet til. Udvid funktionen `build_connections()` i town.js, så den returnerer et objekt hvor hver nøgle er en lokation og værdien er et array som indeholder alle steder som nøglelokationen er forbundet til. Se nedenstående eksempel for et forventet output med nøglen A:

```
const connections = build_connections();
console.log(connections.A);
// Output: Array(3) [ "B", "C", "P" ]
```

- Status for simuleringen kan oprettes som et objekt. Hvis dette status-objekt indeholder information om robottens placering, samt pakkernes nuværende lokation og deres destination, kan der laves et nyt opdateret objekt hver gang robotten flytter sig. Dermed kan man holde styr på overordnet status i simuleringen. Robottens placering kan bare være bogstavet for den tilsvarende lokation. Der er dog flere pakker, så man kunne fx bruge følgende struktur for pakker:

```
let packages = [{ current: "A", destination: "B" }, { current: "A",
destination: "C" }, { current: "A", destination: "G" }];
// 3 pakker der alle er i punktet A og skal leveres til hhv. B, C og G
```

    - I town.js er der givet en skabelon for objektet `state`, der skal initialiseres vha. metoden `init()`. Dette kan bruges som udgangspunkt til et status-objekt. Metoden `move(to)` skal udvides, så det kan bruges til at flytte robotten til punktet angivet i `to`. Der skal tages højde for følgende (implementer det evt. lidt ad gangen):
        - Robotten skal kun flyttes hvis den nuværende position er forbundet til `to`.
        - Hvis der er pakker hvor robotten kommer fra, så skal de flyttes med robotten (den har samlet dem op).
        - Hvis nogle pakker har nået deres destination, så skal de fjernes fra pakkelisten.
        - Metoden skal returnere et nyt status-objekt hvis robotten har flyttet sig.
    - Test funktionaliteten. Afprøv fx nedenstående (overvej selv hvad output skal være) og tilføj egne tests.

```
let test = Object.create(state);
test.init("A", [{ current: "A", destination: "B" }, { current: "A",
destination: "C" }]);
let travel = ["B", "A", "C", "P"];
for (let to of travel) {
    test = test.move(to);
    console.log(test);
}
```

- Logikken til en pakkerobot skal implementeres. Som udgangspunkt kan det afprøves at styre robotten ved at vælge et tilfældigt punkt der er forbundet til den nuværende lokation. Implementer en funktion `randomRobot(state)` som modtager et status-objekt og vælger en tilfældig destination. Funktionen skal returnerer et objekt, som fx `{to: "A"}` (andre robotlogikker vil have flere elementer i objektet de returnerer). Test robotten!

- Implementer en funktion `runRobot(state, robot, memory, i = 0)`, som står for at afvikle simuleringen. Den skal modtage et status-objekt, navnet på en robot funktion (fx. randomRobot), evt. hukommelse der skal videregives til robotfunktionen og tallet i der står for antallet af iterationer. Funktionen skal kalde robotfunktionen for at få næste destination, derefter kalde status-objektet med den givne destination og til sidst kalde sig selv for at afvikle en omgang mere såfremt der stadig er pakker i byen. - Hvis man skal nå at se visualiseringer af robottens færden (hvis det er implementeret), kan man benytte `setTimeout()` til at kalde runRobot.

- **Ekstra:** Tilføj en metode til status-objektet der initialiserer objektet med tilfældig startposition for robot, samt pakker med tilfælde afsender og leveringsadresse.
- **Ekstra:** Opret robotlogikker der leverer pakkerne hurtigere end en tilfældig robot. Se fx Mail Truck Route, Pathfinding og Robot Efficiency.
- **Ekstra:** Opret en funktion eller objekt der kan bruges til at sammenligne effektiviteten af robotter. Se evt. Measuring a robot.

Flere udvidelser

- **Robotten kan kun bære begrænset kapacitet:** Hvordan kan det simuleres at robotten maximalt kan bære fx 2 pakker ad gangen? Hvordan skal automatiseringen justeres?
- **Vægtet graf:** Hvad hvis nogle forbindelser tager længere tid at rejse end andre? Der kan tilføjes omkostninger til hver forbindelser for at danne en såkaldt vægtet graf. Som udgangspunkt kunne disse vægte være afstanden i pixels mellem 2 forbundne lokationer.
- **Flyvende robot:** Hvordan kan simuleringen ændres hvis robotten kan nå alle lokationer i fugleflugt?
- **Design:** Hjemmesiden kan gøres pænere og flere information kan udskrives til hjemmesiden i stedet for konsollen. Der kan tilføjes flere animationer.
- **Flere robotter:** Hvordan kan der tilføjes flere pakkerobotter og hvordan skal de styres?
- **Andre byer:** Afprøv simuleringen med andre placeringer og forbindelser imellem dem. Opstår der uventede problemer?
- **Genetisk programmering:** Robotten kan "lære" den optimale rute selv ved fx genetisk programmering.